Trick Modes in GStreamer

GStreamer Conference 2014, Düsseldorf
17 October 2014

Sebastian Dröge <sebastian@centricular.com>
Centricular Ltd
Who is speaking?

• Sebastian Dröge, long-time GStreamer core developer
• probably touched every piece of code by now
• worked on GStreamer for various companies, now at Centricular
What is this about?

- trick modes
  - slower or faster than real-time playback
  - reverse playback

- case studies: local files, RTSP, HTTP adaptive streaming, DLNA

- theory: how to implement this with GStreamer, how does it work?
Case Study 1: Local Files

- the base case
- we can do random access to every possible position
- assume a container format that knows position of keyframes
  - e.g. MP4, Matroska, not MPEG TS
  - for simplicity, need extra tricks for others
Forward Playback

- intuition: only need to play everything faster or slower
- there should be nothing special needed by any elements other than those who synchronize to the clock: sinks
The SEEK & SEGMENT Event

- Rate changes are triggered with the SEEK event
  - Other fields: format, start, stop, ...

- Element driving the pipeline has to tell downstream about position and synchronization information: SEGMENT event
  - Format, start, stop, time, base fields
  - Rate field

- Used to convert buffer timestamps to different times
  - Running time (→ synchronization)
  - Stream time (→ position reporting)
Recap: Times in GStreamer

Stream time

Running time

Clock time

100 ms stream

base time
Picture of times in GStreamer
why so complicated: looping and stuff we mention later
How does it work?

• video sinks just adjust frame durations

• audio sinks have to resample
  ◦ base class does that already

• every other element just forwards rate information and timestamps as is

→ synchronization happens twice as fast
→ stream time is reported as is
Reverse Playback

• forward was easy, what about reverse?
• intuition: render everything backwards and handle speed differences as before
The SEEK & SEGMENT Event

• same as before but rate < 0.0
  ° start < stop as before! but playback from stop to start

• running time must not go backward, stream time has to
  ° different formulas for forward and backward
But compressed data can't be sent in reverse

- Keyframes in video
- Audio frames contain many samples that need to be reversed
Mode of Operation in Elements

File with 5 GOPs

ABCDE  FGHIJ  KLMNO  PQRST  UVWXZ

demuxer

UVWXZ  PQRST  KLMNO  FGHIJ  ABCDE

GOP with 5 frames

U  V  W  X  Y

decoder

Y  X  W  V  U
reordering picture
The STEP Event

• for stepping a specific amount
  ◦ format, amount, rate, flush fields

• allows changing rate but not direction
  ◦ without flushing and immediately
  ◦ can be handled only in sink, nothing else needs to know
Solves

• "perfect" forward and reverse playback
  ◦ no frame is lost and everything is in order
  ◦ good for e.g. video editing
Problems?

- complicated in demuxers
  - not fully implemented everywhere yet
  - difficult in case of e.g. MPEG TS

- 32x data rate of 32x playback
  - might be too much for the CPU or hardware codecs or also just for reading the data

- high memory pressure for reverse playback
  - complete raw GOP in memory

- requires efficient random access
Status

• forward trick modes should work in all demuxers
  ◦ not only with local files

• reverse trick modes implemented in MP4, Matroska, Ogg and AVI demuxer

• parser, decoder and sink base classes handle it

• generally works well
Application Side Trick Modes

• so far very heavy performance requirements
• let's take a step back
• how would we implement trick modes from the application?
Flushed Seek in PAUSED

- seek to the start position
  - use KEYUNIT and SNAP_BEFORE/SNAP_AFTER seek flags
- wait until seek is done
- calculate time taken
- based on the rate select next seek position
- repeat
Properties

- works with all demuxers out of the box
- automatically adapts to delays caused by seeking, etc
Problems?

- needs to be implemented in every application
- not exactly trivial to implement but it works with every element that allows seeking
- no knowledge about keyframes positions
  - could play the same segment multiple times
SKIP Mode

- solve these problems by moving logic to demuxers
  - under discussion
  - basically play and skip in the demuxer
The SEEK & SEGMENT Event

- seek event as before but with SKIP flag
  - rate $\neq 1.0$

- multiple, skipping, rate=1.0 segment events
  - same as with application-side seeks
Possible future improvements

- I/B-frame skipping, disable audio/subtitles, ...
  - needs further seek flags

- Automatic adjustments to seek delays via QoS events
Solves

• input bandwidth / datarate limitations
  ◦ if implemented properly in the demuxer

• processing constraints in the decoders and renderers
Problems?

- no "perfect" trick modes
- keyframe positions are not always known (e.g. MPEG TS)
- potentially a lot of unnecessary parsing in the demuxers
  - would also cause high input bandwidth requirements
Status

- application-side should work with every pipeline
- demuxer-side has to be implemented still
  - only design discussions so far: Bugzilla #735666
What about remote content?

• Clearly we can't just stream stuff e.g. 32x faster
• Knowledge about keyframe positions might not exist
• Random access might be slow
Case Study 2: RTSP

- HTTP-style protocol for setting up (mainly) RTP sessions
- control flow via RTSP, data flow via RTP
  - stream and parameter discovery
  - stream selection, play/pause, seeking, ...
- low-latency streaming
RTSP Trick Mode Support

- server-side playback rate adjustments
  - server transcodes as required and possible
  - returns stream with closest possible scale
  - e.g. stream with half duration for rate 2.0
  - "perfect" trick modes

- time based seeks
  - efficient SKIP mode

- also a speed parameter for just sending data slower/faster
  - RTP sent in real time
The SEEK & SEGMENT Event

- SEEK as before, handled by rtspsrc
- SEGMENT event special
  - new applied_rate field for server side changes
  - e.g. rate=1.0, applied_rate=2.0
  - stream time scaled instead of running time
Solves

• when done server side without speed parameter
  ◦ input bandwidth / datarate limitations
  ◦ no unneeded parsing and processing
  ◦ processing constraints in the decoders and renderers
  ◦ "perfect" trick modes

• everything can be done on the server
Problems?

• what if not supported by server or only specific rates?
  ◦ combination of different modes

• not fully implemented in GStreamer and many other implementations

• not supported by many servers

• potentially heavy load on the server
Status

- RTSP source supports forward trick modes via Speed and Scale
  - reverse should work but is untested due to lack of a server that supports it
- RTSP server only supports sending faster/slower
  - reverse not implemented yet
  - no transcoding yet
Case Study 3: HTTP Adaptive Streaming

- many standards: HLS, DASH, Smooth Streaming, ...
  - DASH most complicated but biggest support in the industry

- basically
  - a manifest / playlist with stream information and locations
  - stream variants split into fragments
  - download fragments and play them as one combined stream
Advantages over progressive HTTP Streaming

- allows selection of bitrates, codecs, resolutions, languages, ...
  - just place variants into a different set of fragments
  - seamless switching during playback
- easy seeking on fragment boundaries
- simple high-latency live streaming
HTTP Adaptive Streaming Trick Modes

• combination of what we had so far
  ◦ client-side (rate changes, SKIP, ...) with the known problems
  ◦ additional optional features

• I-frame only variants / codingDependency=false sub-representations (HLS/DASH)

• lower quality variants / sub-representations (HLS/DASH)
  ◦ codec complexity, bitrate, ...
  ◦ lower framerate like server-side transcoding
HTTP Adaptive Streaming Trick Modes

- separately stored I/P/B frame positions (sidx/ssix) (DASH)
  - allows efficient SKIP mode

- information about max. rate without increasing codec complexity (DASH)
  - i.e. with staying in the same codec level
Problems?

• often none of these extra features used unfortunately
  ◦ heuristics, assuming there's a keyframe at the beginning of a fragment, ...

• how to find and / or forward keyframe positions
  ◦ demuxer knows container format, adaptive streaming demuxer doesn't
  ◦ parsing of parts of container format?
Status

• HLS, DASH and Smooth Streaming are supported in general
• HLS I-frame playlists are supported
• seeking supported and normal client-side trick modes
• more work needed for
  ◦ proper stream selection (quality & e.g. language)
  ◦ support of trick mode specific DASH features
Case Study 4: DLNA

- Digital Living Network Alliance

- lots of guidelines and specifications for interoperability of media devices, based on UPnP
  - complicated and huge

- for our purposes here
  - HTTP-like protocol with custom HTTP headers
  - reusing the http URI scheme
How to implement in GStreamer

• due to using http URI scheme and requiring extended HTTP protocol, architecture not trivial
  ◦ no implementation from the GStreamer project available yet

• a few ideas, talk to me later
DLNA Trick Modes

- like RTSP

- server-side playback rate adjustments
  - "perfect" trick modes

- time based seeks
  - efficient SKIP mode

- byte based seeks
  - can be used like normal client-side trick modes and SKIP mode
  - standard HTTP
  - can use downloadbuffer (queue2) element
Problems?

- highly depends on which features are supported by the server
- not very efficient with byte based seeks
Status

• no implementation from the GStreamer project yet
  ◦ some work happening in that area

• all the requirements are there except for DLNA-HTTP support

• requires a few fixes in demuxers
Summary

• all infrastructure is there thanks to our sophisticated synchronization model
  ◦ see gstreamer/docs/design for details and formulas

• implementations for various kinds of trick modes too
  ◦ but also many are lacking still and work is happening around these areas

• same story with support for different streaming protocols

• stabilization required in various places
  ◦ gst-validate!

• expect great things to happen in the future
Questions?
also feel free to talk to me later or write a mail
sebastian@centricular.com
summary of this talk will be on my personal blog
https://coaxion.net/blog
Thank You!

Pictures

https://flic.kr/p/a3hrYe
https://flic.kr/p/J9hJ1
https://flic.kr/p/dZRG7K
https://flic.kr/p/bkgRhN
https://flic.kr/p/9BM24Q
https://flic.kr/p/5BWVUb
https://flic.kr/p/7NWqwX
https://flic.kr/p/632Ye5